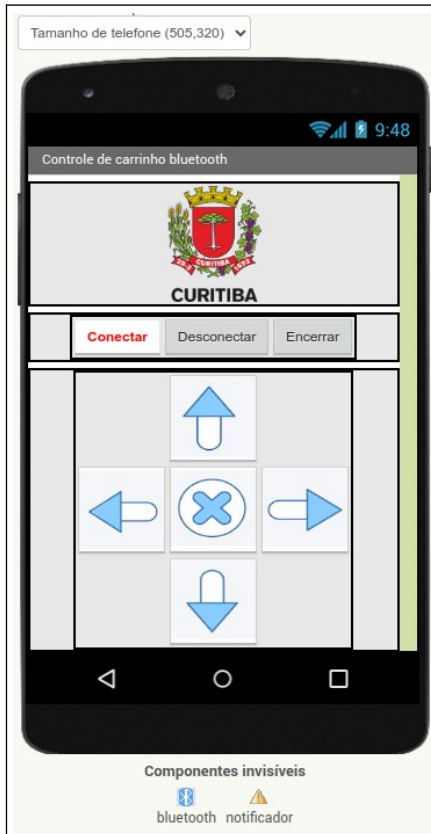


# Controle de carrinho por App e *bluetooth*

## Visão do App



O App será desenvolvido por meio do MIT App Inventor.

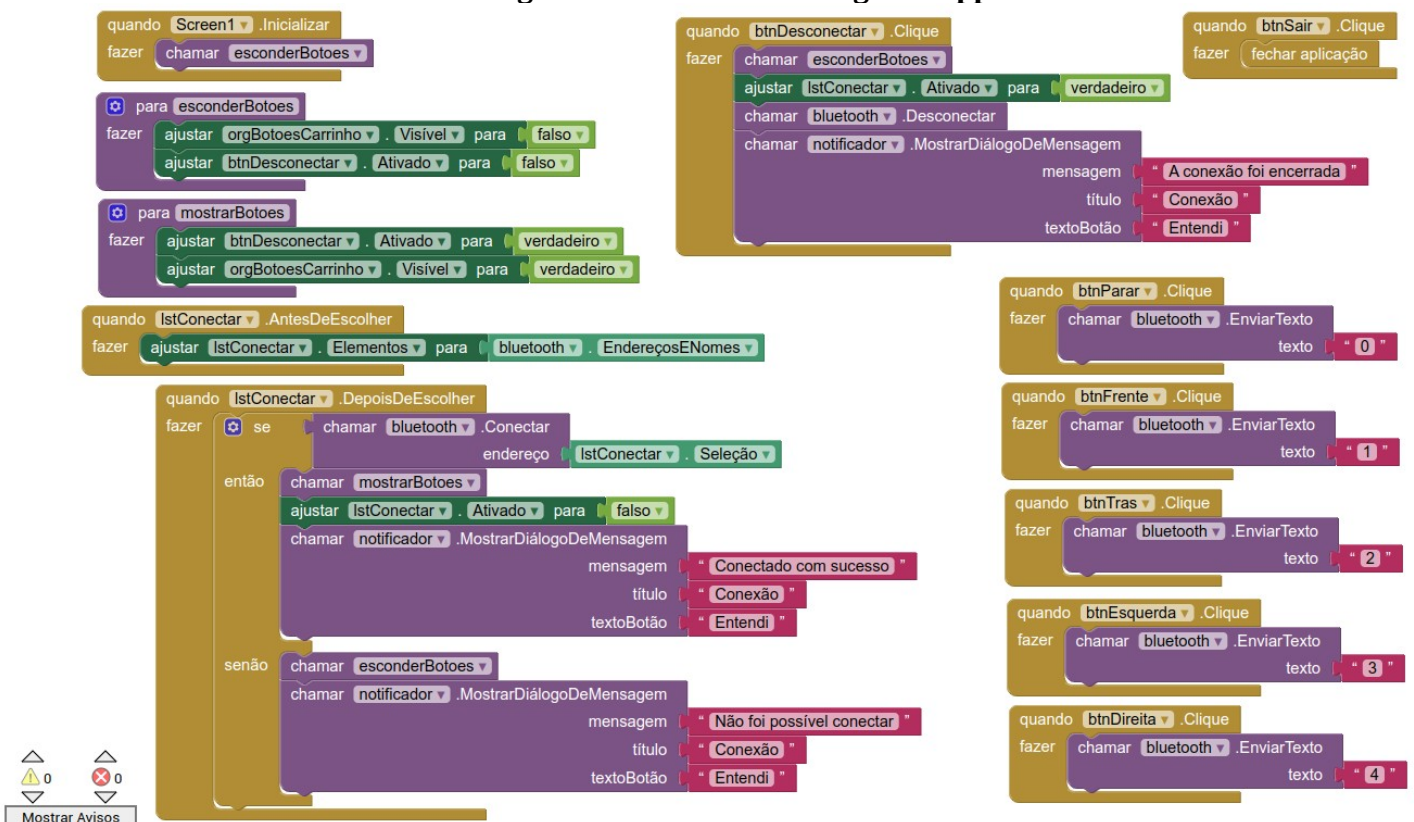
Possui uma tela como a exibida ao lado.

Seus controles estarão divididos em dois blocos:

- Controles do App
  - Conectar ao *bluetooth* do Arduino
  - Desconectar do *bluetooth* do Arduino
  - Encerrar (fechar) o App
- Controles do carrinho
  - Enviar código de seguir em frente (↑)
  - Enviar código de seguir em voltar/ ir para trás (↓)
  - Enviar código para seguir à esquerda (←)
  - Enviar código para seguir à direita (→)
  - Enviar código para parar os motores (X)

NOTA: para que o App funcione do ponto de vista de conexão *bluetooth* é necessário que o módulo *bluetooth* tenha sido previamente pareado utilizando-se as funções de conexão do celular ou tablet.

## Visão global dos blocos de código do App





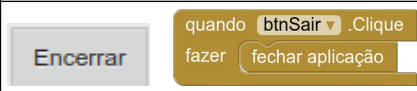
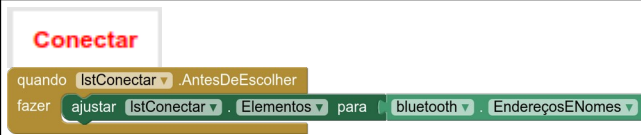
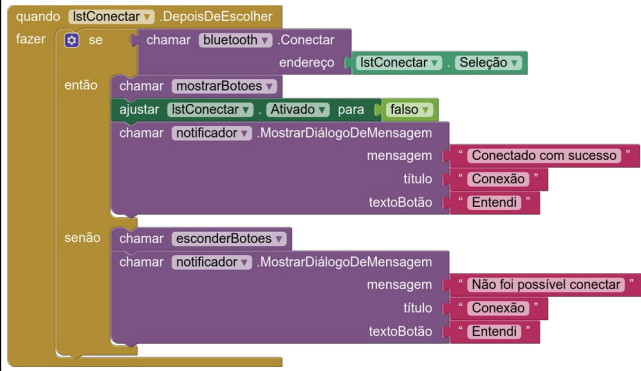
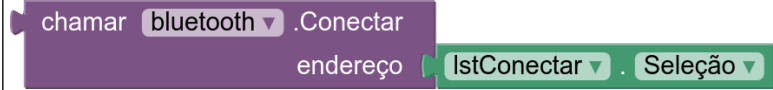


## Resumo do funcionamento:

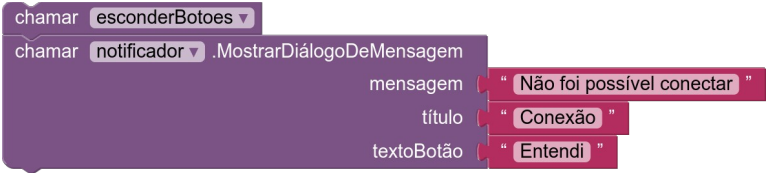
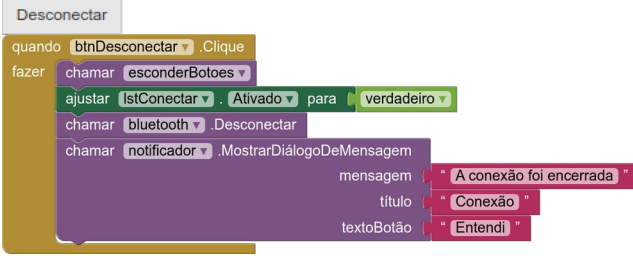
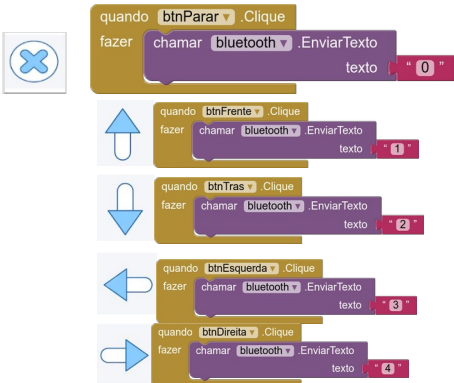
1. ao ser iniciado, o App esconde os botões de controle do carrinho e o de desconexão;
2. quando for acionado o botão 'Encerrar' ele termina;
3. quando for acionado o botão 'Conectar', será exibida uma lista de dispositivos bluetooth conhecidos pelo celular e pareados, e, se um dispositivo válido for selecionado, haverá conexão, caso em que o

botão ‘Desconectar’ ficará ativo, o botão ‘Conectar’ inativo e os botões de controle do carrinho ficarão visíveis;

4. cada vez que um botão de controle for acionado, ele enviará um código, sendo que o botão parar (X) envia ‘0’, o botão para frente (↑) envia ‘1’, o botão para trás (↓) envia ‘2’, o botão para esquerda (←) envia ‘3’, e o botão para direita (→) envia ‘4’ (estes números podem ser modificados à vontade, mas lembre-se de que no código do Arduino também devem ser realizados ajustes correspondentes).

### Explicação dos blocos de código


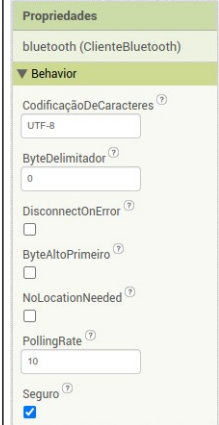
|  |  |
|--|--|
|    | <p>Quando a tela inicial (<i>Screen1</i>) inicializar (ou seja, quando o App for acionado e iniciar), chama a função ‘esconderBotoes’</p>  |
|    | <p>Quando esta função for chamada, ela irá ajustar a propriedade ‘Visível’ da barra de organização horizontal ‘orgBotoesCarrinho’ para ‘falso’ – isto fará com que a barra (e seu conteúdo, que é uma tabela com os cinco botões de controle), não sejam visíveis – e, portanto, fará com que o usuário não possa acionar os botões de controle do carrinho pois não há nenhum carrinho concetado. Também ajustará a propriedade ‘Ativado’ do botão ‘btnDesconectar’ para falso (pois se não há conexão ainda não faz sentido desconectar, evitando erros de operação – o botão ficará visível mas não poderá ser acionado).</p>   |
|  | <p>A qualquer momento que o botão ‘Encerrar’ for acionado a aplicação termina e sai da memória.</p>  |
|  | <p>Quando o botão de lista ‘lstConectar’ for acionado, ele prepara uma lista com todos os dispositivos <i>bluetooth</i> conhecidos e pareados para que o usuário possa selecionar por meio do endereço universal ‘MAC (Media Access Control)’ ou do nome do dispositivo.</p>   |
|  | <p>Após a seleção de um dispositivo da lista será chamada a conexão <i>bluetooth</i>:</p>  <p><b>SE</b> a conexão der certo, <b>ENTÃO</b></p>  <p>Será chamada a função ‘mostrarBotoes’:</p>  <p>Esta função mudará a propriedade ‘Ativado’ do botão de Desconexão e também deixará visíveis os notões de controle de direção. O botão de conexão ficará invisível. Será chamado o notificador que exibirá uma mensagem (caixa de texto) informando que a conexão com o dispositivo selecionado foi bem sucedida</p> |


|   |  |
|---|--|
|   | <p><b>SENÃO</b></p>  <p>Caso a conexão não tenha dado certo será chamado o procedimento ‘esconderBotoes’, já comentado na inicialização do App</p>   |
|   | <p>Quando o botão ‘Desconectar’ for acionado será chamado o procedimento ‘esconderBotoes’ para que não possam ser acionados os botões de direção; será ajustada a propriedade de ativação da lista ‘lstConectar’ para que possa ser realizada conexão com outro dispositivo; o <i>bluetooth</i> será desconectado, interrompendo o fluxo de dados; e, será chamada uma caixa de texto que informará o usuário que a conexão com o dispositivo foi encerrada.</p> |
|  | <p>Quando o botão ‘Parar (X)’ for acionado, será enviado ao bluetooth conectado o valor numérico ‘0’ (zero).</p> <p>O mesmo ocorrerá para quando forem acionados os demais botões de controle de direção, com mudança somente do valor enviado.</p> <p>O dispositivo receptor deverá realizar o tratamento do valor recebido. - em nosso caso, o Arduino, veja o código na sequência das explicações.</p>  |

### Componentes não visíveis presentes no App



Estes componentes são utilizados pelo App porém não ficam visíveis na tela, somente estão disponíveis nos blocos de programação.

|  |   |   |
|--|---|---|
|  |  | <p>Este é o componente responsável pela comunicação por meio do protocolo <i>bluetooth</i>.</p> <p>Dentre suas propriedade é importante verificar o ‘ByetDelimitador’, que deve ser ‘0’, e o ‘PollingRate’, para o qual o valor utilizado foi 10.</p> |
|--|---|---|

|  |   |
|--|---|
| <br><b>notificador</b> | <div> Propriedades </div> <div> notificador (Notificador) </div> <div> ▼ Appearance </div> <div> CorDeFundo <sup>(?)</sup><br/> <input checked="" type="checkbox"/> Padrão </div> <div> TamanhoDoNotificador <sup>(?)</sup><br/> Longo ▼ </div> <div> CorDeTexto <sup>(?)</sup><br/> <input type="checkbox"/> Padrão </div> |
|--|---|

## Comportamento do Arduino

O Arduino deverá testar o valor recebido pelo modem *bluetooth* e responder com uma ação correspondente. Verifique no restante do material a necessidade de conexão elétrica entre os dispositivos, e quais foram os pinos de controle utilizados para, se for o caso, ajustar o código.

As conexões são identificadas no início do programa:

```

3 int ControleEsquerdo = 3; //Pino 9 do L293D
4 int ControleDireito = 5; //Pino 1 do L293D
5 int EsquerdoA = 6; //Pino 10 do L293D
6 int EsquerdoB = 7; //Pino 15 do L293D
7 int DireitoA = 8; //Pino 7 do L293D
8 int DireitoB = 9; //Pino 2 do L293D
9 int EnergiaEsquerdo = 100;
10 int EnergiaDireito = 100;

```

As declarações das linhas 3 e 4 referem-se ao controle (PWM) dos motores esquerdo e direito; as observações indicam em qual pino do L293 ligar a porta do Arduino declarada – você pode mudar as portas, basta ajustar as ligações.

As linhas 5 e 6 são para controle da direção de rotação do motor ligado ao lado esquerdo e as linhas 7 e 8 para controle de rotação do motor direito.

As linhas 9 e 10 correspondem ao valor que será passado para escrita analógica PWM e irão variar ao longo da execução do programa. O valor proposto para início é 100, que dará uma velocidade média.

Na sequência, temos a inicialização do Arduino:

```

12 void setup( )
13 {
14     Serial.begin(9600);
15     pinMode(ControleEsquerdo, OUTPUT);
16     pinMode(ControleDireito, OUTPUT);
17     pinMode(EsquerdoA, OUTPUT);
18     pinMode(EsquerdoB, OUTPUT);
19     pinMode(DireitoA, OUTPUT);
20     pinMode(DireitoB, OUTPUT);
21     analogWrite(ControleEsquerdo, EnergiaEsquerdo);
22     analogWrite(ControleDireito, EnergiaDireito);
23 }

```

A declaração da linha 14 foi colocada para permitir controle dos motores por meio de interação na interface serial.

Entre as linhas 16 e 20 temos a declaração de que as portas correspondes serão saídas.

As linhas 21 e 22 escrevem o valor (inicialmente 100) que corresponderá à potência PWM nos motores.

Foi escrito um procedimento para parar os motores; isto evita que a energia seja revertida bruscamente:



```

25 void pararMotores() {
26     digitalWrite(EsquerdoA, LOW);
27     digitalWrite(EsquerdoB, LOW);
28     digitalWrite(DireitoA, LOW);
29     digitalWrite(DireitoB, LOW);
30     delay(100);
31 }

```

As linhas 26 e 27 colocam a rotação do motor esquerdo em sentido desligado e as linhas 28 e 29 fazem o mesmo com o motor direito.

A linha 30 foi incluída para que exista uma pequena pausa entre os novos controles de direção, de forma a evitar mudanças muito violentas na rotação.

A diferença entre ir ‘para frente’ ou ‘para trás’ reside no sentido de movimento do motor. Foram escritas duas funções que fazem com que os motores girem em sentidos opostos (obviamente conforme for montado o motor no carrinho os nomes poderão estar invertidos).

```

33 void emFrente(){
34     digitalWrite(EsquerdoA, HIGH);
35     digitalWrite(EsquerdoB, LOW);
36     digitalWrite(DireitoA, HIGH);
37     digitalWrite(DireitoB, LOW);
38 }
39
40 void reverso(){
41     digitalWrite(EsquerdoA, LOW);
42     digitalWrite(EsquerdoB, HIGH);
43     digitalWrite(DireitoA, LOW);
44     digitalWrite(DireitoB, HIGH);
45 }

```

Note que a direção de movimento é definida pelos valores lógicos aplicados: HIGH e LOW para um sentido e LOW e HIGH para o sentido inverso (LOW e LOW para o motor; e, HIGH e HIGH não é um valor válido). Note a diferença: por exemplo, para o motor esquerdo o sentido (frente) é EsquerdoA = HIGH e EsquerdoB = LOW; para (reverso) é EsquerdoA = LOW e EsquerdoB = HIGH.

Finalmente, o *loop* principal:

```

47 void loop( )
48 {
49     if(Serial.available() > 0) {
50         int val = Serial.parseInt();
51         switch (val){

```

A linha 49 verifica se há dados na interface serial, e se houver a linha 50 fará a leitura de um valor inteiro. A linha 51 compara este valor lido verificando na sequência qual o valor correspondente.

```

52         case 0:
53             Serial.println("Parado"); //esta linha pode ser comentada ou suprimida para uso com bluetoo
54             pararMotores();
55             break;

```

Se o valor for zero mostra o comando na interface serial (linha 53) e chama, na linha 54, a função de parar os motores.

```

56         case 1:
57             Serial.println("Frente"); //esta linha pode ser comentada ou suprimida para uso com bluetoo
58             pararMotores();
59             emFrente();
60             break;

```

Na linha 56 compara o valor lido com '1' e, se for válido, escreve o comando na interface serial (linha 57) e para os motores (linha 58). Em seguida, comanda os motores (linha 59): ambos irão girar no mesmo sentido (se não for o caso, você pode inverter os fios dos motores em seu protótipo ou os fios que ligam o Arduino no L293 – fios A e B de cada motor).

```
61         case 2:
62             Serial.println("Tras"); //esta linha pode ser comentada ou suprimida para uso com bluetooth
63             pararMotores();
64             reverso();|
65             break;
```

Caso o valor lido na serial ou recebido por bluetooth for 2, a mensagem do comando é exibida (linha 62), os motores são parados (linha 63) e em seguida são ativados em sentido inverso de movimento (linha 64).

Vejamos agora como fazer uma curva. Podemos fazer isto mantendo o motor de um dos lados funcionando e alterando o motor do lado para o qual queremos virar: (a) parando-o, (b) fazendo-o girar em sentido contrário ou (c) diminuindo sua velocidade. Você pode testar as três alternativas em seu protótipo: a primeira (a) causa uma mudança brusca, a segunda (b) uma mudança mais brusca ainda e a terceira (c) faz uma curva que vai de suave a violenta, dependendo das velocidades escolhidas. Esta terceira pode ser usada para realização de curvas de raio maior. No nosso código esta é a que foi escolhida para início dos ensaios.

```
66         case 3:
67             Serial.println("Curva à esquerda"); //esta linha pode ser comentada ou suprimida para uso c
68             pararMotores();
69             EnergiaEsquerdo -= 10;
70             if (EnergiaEsquerdo < 10){
71                 EnergiaEsquerdo = 10;
72             }
73             analogWrite(ControleEsquerdo, EnergiaEsquerdo);
74             EnergiaDireito += 10;
75             if (EnergiaDireito > 250){
76                 EnergiaDireito = 250;
77             }
78             analogWrite(ControleDireito, EnergiaDireito);
79             emFrente();
80             break;
```

O procedimento para realizar uma curva à esquerda adotado é: diminuir a rotação do motor esquerdo e aumentar a rotação do motor direito. Na linha 69 o valor atual de energia do motor esquerdo é diminuído de 10 (a declaração “EnergiaEsquerdo -= 10” faz com que o valor da variável de controle de potência do motor esquerdo seja diminuída de 10). Na primeira execução o valor inicial (definido no início do programa) será 100; o valor aplicado ao motor será  $100 - 10 = 90$ . Se o mesmo comando for repetido, sem que outro comando que varie a potência tenha sido executado (curva à direita), no próximo momento o valor será  $90 - 10 = 80$ , depois  $80 - 10 = 70$  e assim sucessivamente (com um teste na linha 70 que garante que o valor mínimo seja 10).

Já no caso do motor direito o valor será aumentado de 10 em 10 (linha 74, “EnergiaDireito += 10”), até um máximo de 250 (linhas 75 e 76).

Após definidos os valores de energia os motores são acionados (linha 79).

Note que o motor esquerdo vai girar com potência menor (90) do que o direito (110) de forma que haverá tendência de que o lado esquerdo ‘trave’ o movimento, provocando uma curva à esquerda (os valores e suas variações devem ser testados em seu protótipo de forma a conseguir os valores ótimos para as curvas desejadas).

```
81         case 4:
82             Serial.println("Curva à direita"); //esta linha pode ser comentada ou suprimida para uso co
83             pararMotores();
84             EnergiaDireito -= 10;
85             if (EnergiaDireito < 10){
86                 EnergiaDireito = 10;
87             }
88             analogWrite(ControleDireito, EnergiaDireito);
89             EnergiaEsquerdo += 10;
90             if (EnergiaEsquerdo > 250){
91                 EnergiaEsquerdo = 250;
92             }
93             analogWrite(ControleEsquerdo, EnergiaEsquerdo);
94             emFrente();
95             break;
96     }
```

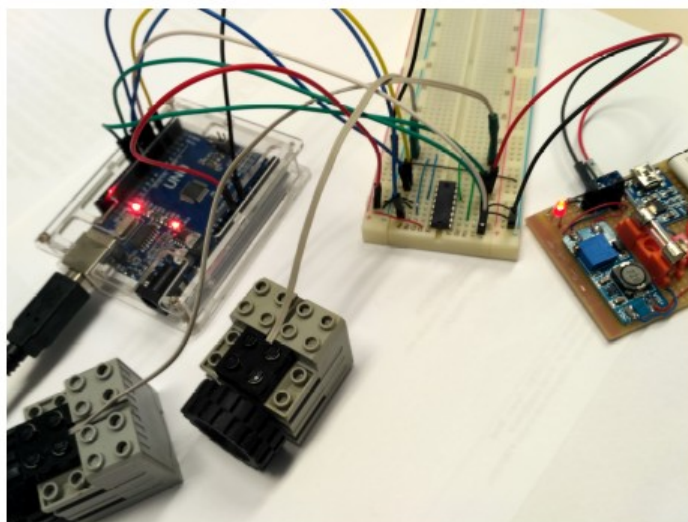
Entre as linhas 81 e 95 ocorre o mesmo processo descrito para curva à esquerda, porém agora realizando uma curva à direita ( o motor do lado diretito vai girar mais devagar que o motor esquerdo).

**OBSERVAÇÃO** – as variáveis que controlam a potência não serão restabelecidas ao valor original (100). Se for realizado um comando de curva à esquerda e na sequência mais dois comandos idênticos a curva deve ficar cada vez mais acentuada (90 à esquerda e 110 à direita no primeiro comando, respectivamente 80 e 120 e depois 70 e 130 nos comandos seguintes). O valor originalmente estabelecido no início do programa (100), ou qualquer outro, poderia ser controlado por um outro comando (por exemplo, opção ‘5’ para reiniciar os valores em 100; opção ‘6’ para colocar o valor em 200; opção ‘7’ para colocar o valor em 50; infomrar o valor diretamente...) - você pode testar isso.

### Ligações do Arduino e motores



Pinos GND = 4, 5, 12, 13  
Positivo +5V = 16  
Motor 'A' = 3, 6  
Motor 'B' = 11, 14  
Energia mot.= GND no GND  
+9V no pino 8  
Controle 'A' = 2,7  
Controle 'B' = 10, 15  
Potência 'A' = 1  
Potência 'B' = 9



#### Controle 'A'

2 do L293 = 9 Arduino

7 do L293 = 8 Arduino

#### Controle 'B'

10 do L293 = 6 Arduino

15 do L293 = 7 Arduino

#### Potência 'A'

1 do L293 = 5 Arduino

#### Potência 'B'

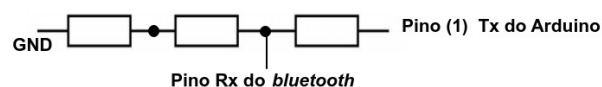
9 do L293 = 3 Arduino

(Pinos 3 e 5 são PWM)

No meu experimento foram utilizados motores do kit LEGO RCX, os quais funcionam com 9Volts. Foi utilizada uma bateria de íons de lítio de celular antigo (3,7Volts) e um conversor tipo 'step up' para subir esta tensão de forma a alimentar os motores com energia suficiente – a alimentação do circuito integrado L293 é fornecida pelo Arduino (já que ele funciona com 5Volts e uma corrente baixa).

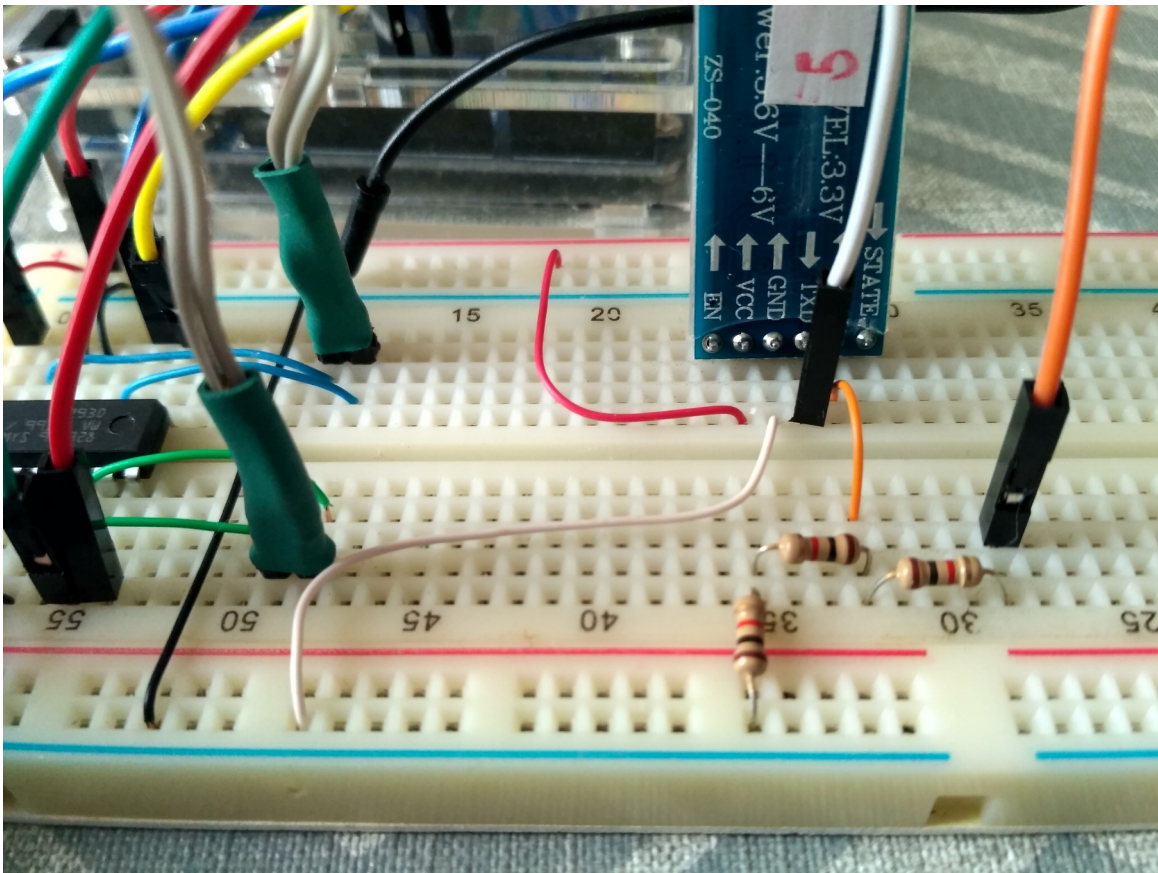
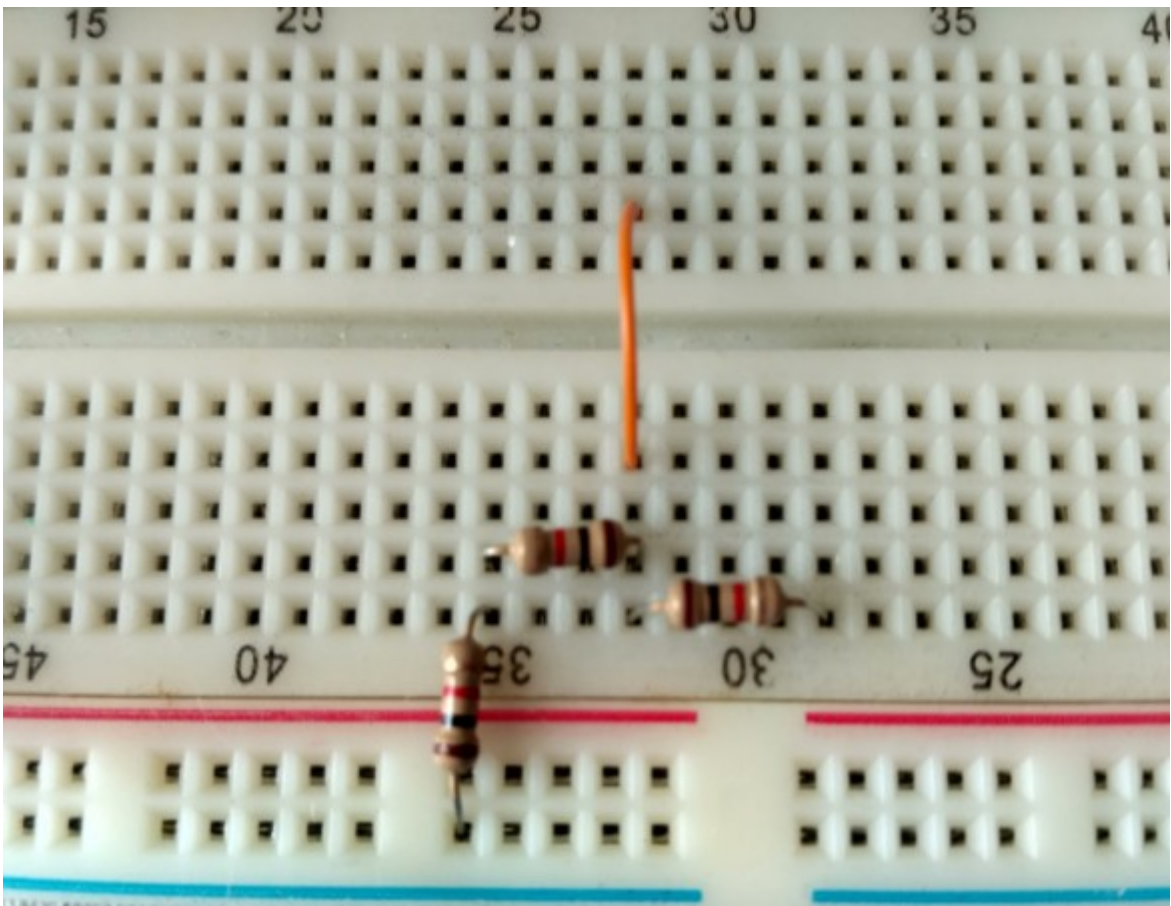
Na sequência de seus experimentos você pode comentar as linhas 42, 45, 52, 59 e 76 (relativas à escritas na interface serial), pois elas são somente para testes. Depois, conecte um modem *bluetooth* nas portas '0' (Rx) e '1' (Tx) do Arduino e, sem nenhuma modificação no código, pode controlar o carrinho pelo App. O modem *bluetooth* se comportará como a interface serial e os comandos recebidos do App (números 0 a 4, conforme o caso) irão controlar o código conforme visto nas explicações anteriores.

Lembre-se de que o modem *bluetooth* tem o canal de recepção ajustado para um máximo de 3,3Volts e é necessário reduzir a tensão (de 5Volts) enviada pelo Arduino. Um exemplo é:



(os resistores do exemplo são de 1000 ohms, você pode usar outros valores a partir de 220 ohms, desde que os três sejam exatamente do mesmo valor).





ATENÇÃO – este tipo de ligação é simplificado e com resultado garantido porém o modem bluetooth não pode estar conectado quando for transferido código pela interface USB pois a conexão interna é a mesma!